

Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Zespół:
Kot Jarosław
Prusicki Jakub

Łamacz haseł

Projekt zespołowy
na studiach stacjonarnych
o kierunku **Informatyka**
Stopień II

Opiekun projektu:
dr inż. Paweł Paduch

Kielce, 2023

SPIS TREŚCI

1. Metoda słownikowa - opis algorytmu.	3
2. Haszowanie.	4
2.1. Co to jest hash ?	4
2.2. MD5.	4
2.3. SHA-1,SHA-256.	5
2.4. Haszowanie "Solone".	8
3. Rozwój projektu.	8
3.1. Wydajność Brute Force.	8
3.2. Generowanie możliwych haseł na podstawie słownika.	9
3.3. Wyliczanie zakresów.	10
3.4. Popularne hasła.	11

1. METODA SŁOWNIKOWA- OPIS ALGORYTMU.

Metoda słownikowa to jedna z najprostszych i najskuteczniejszych metod łamania haseł. Polega ona na próbie odgadnięcia hasła za pomocą słownika zawierającego różne kombinacje słów, haseł i znaków specjalnych. Istnieją różne wersje metody słownikowej, ale podstawowa koncepcja jest taka sama. Atakujący używa listy haseł, która zawiera popularne hasła, słowa i kombinacje znaków. Jeśli hasło znajduje się na liście, atakujący może uzyskać dostęp do chronionej informacji. Jedną z popularnych wersji metody słownikowej jest tzw. "attack by list". W tej metodzie atakujący wykorzystuje gotową listę haseł, która jest zazwyczaj tworzona na podstawie często używanych haseł, słów kluczowych, nazw użytkowników itp. Lista ta może być pobrana z internetu lub stworzona przez samodzielne zbieranie informacji o ofierze. Następnie atakujący przeprowadza atak, używając tych haseł na kolejnych stronach internetowych lub w systemach informatycznych, które próbuje złamać. Inną wersją metody słownikowej jest tzw. "brute force attack". Ta metoda polega na wypróbowaniu wszystkich możliwych kombinacji haseł, aż zostanie znalezione poprawne hasło. To jednak wymaga ogromnej ilości czasu i mocy obliczeniowej, więc jest mało efektywna w przypadku dłuższych i bardziej złożonych haseł. Istnieją również metody hybrydowe, które łączą w sobie metody słownikowe i brute force. Atakujący używa listy haseł, ale dodaje do niej różne kombinacje liter i cyfr, co zwiększa szansę na odgadnięcie hasła. Jednym ze sposobów zabezpieczenia się przed atakami metodą słownikową jest stosowanie silnych haseł. Silne hasło to takie, które składa się z co najmniej ośmiu znaków, zawiera duże i małe litery, cyfry oraz znaki specjalne. Należy również unikać używania popularnych haseł i słów, takich jak "123456" czy "password". Innym sposobem jest stosowanie technik szyfrowania haseł, takich jak funkcje skrótu (hashing) lub algorytmy szyfrowania. Szyfrowanie uniemożliwia odczytanie hasła przez atakującego nawet w przypadku jego przechwycenia. Co zostanie opisane szerzej w punkcie nr 2 (rodzaje haszowań).

2. RODZAJE HASZOWAŃ.

2.1. CO TO JEST HASH ?

Hash'em nazywamy wynik działania operacji matematycznej (nazywanej funkcją skrótu) na określonym ciągu znaków (np. na hasle lub pliku). Funkcja ta przekształca podane przez użytkownika dane wejściowe (np. hasło) na krótką, posiadającą stały rozmiar wartość znakową. Ważną własnością funkcji skrótu jest to, że jest ona nieodwracalna. Osoba mając hash nie może zastosować funkcji odwrotnej do funkcji skrótu by poznać ciągu znaków (np. hasła), dla którego dany hash został wygenerowany. Haszowanie jest to pewna technika rozwiązywania ogólnego problemu słownika. Przez problem słownika rozumiemy tutaj takie zorganizowanie struktur danych i algorytmów, aby można było w miarę efektywnie przechowywać i wyszukiwać elementy należące do pewnego dużego zbioru danych (uniwersum). Przykładem takiego uniwersum mogą być liczby lub napisy (wyrazy) zbudowane z liter jakiegoś alfabetu^[4]. Ogólnie rzecz biorąc hash to wynik zastosowania funkcji skrótu na pewnym ciągu znaków (hasle lub pliku). Hash pozostaje stały dla danego pliku, a każda choćby najmniejsza modyfikacja jednego znaku powoduje niezgodność nowo zmodyfikowanego pliku ze starym hashem.^[1]

Przykład:

Po zastosowaniu funkcji skrótu MD5 na hasle **AlaMaKota1234!** otrzymamy wartość wyjściową w postaci ciągu znaków **e3f52f31c9bd800b9724ced17ba8be96**.

2.2. MD5

Mimo popularności MD5 nie jest już uważane za funkcję w pełni bezpieczną – opracowano wiele technik znajdujących kolizję tej funkcji. Jedną z nich jest metoda *MD5 Tunneling* znajdująca kolizję w czasie poniżej 10 sekund, wykorzystując do tego moc laptopa przeciętnej klasy.

Odkrycia tego rodzaju oczywiście są ciekawe, ale w praktyce nie wpływają znacznie na bezpieczeństwo hashy MD5. Większość technik znajdowania kolizji podaje na wejściu MD5 dane binarne, a więc dane ze zdecydowanie szerszego zakresu niż znaki ASCII.

Chociaż dzisiaj nie istnieją efektywne metody znajdujące kolizje w hashach MD5 haseł, odradza się stosowanie tej funkcji. Istnieją realne przesłanki wskazujące, że sytuacja ta zmieni się w najbliższych latach.

```

public static string CreateMD5(string input)
{
    using (System.Security.Cryptography.MD5 md5 = System.Security.Cryptography.MD5.Create())
    {
        byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(input);
        byte[] hashBytes = md5.ComputeHash(inputBytes);

        //return Convert.ToHexString(hashBytes); // .NET 5 +

        // Convert the byte array to hexadecimal string
        StringBuilder sb = new System.Text.StringBuilder();
        for (int i = 0; i < hashBytes.Length; i++)
        {
            sb.Append(hashBytes[i].ToString("X2"));
        }
        return sb.ToString().ToLower();
    }
}

```

Rys. 2.2.1 MD5.

2.3. SHA.

SHA1 jest obecnie największym konkurentem MD5. Istnieje kilka ataków teoretycznych na tę funkcję, jednak w praktyce nie zagrażają one jeszcze bezpieczeństwu hashy. Jednak tego rodzaju sytuacje sugerują, że warto zainteresować się nowszą wersją algorytmu.

SHA2 jest w tej chwili jedną z najbezpieczniejszych wersji algorytmów rodziny SHA. SHA2 istnieje w czterech odmianach: SHA-224, SHA-256, SHA-384 oraz SHA-512. Wszystkie warianty działają w podobny sposób, jednak używają struktur danych o różnej wielkości oraz zwracają hash różnej długości. Niestety funkcje z tej grupy nie są często stosowane, ponieważ ich użycie często wymaga kompilacji dodatkowych modułów (np. do baz danych). Dodatkowym powodem było też oczekiwanie na finalną wersję algorytmu SHA3, który używa nowego podejścia – więc teoretycznie ataki na wcześniejsze wersje SHA nie powinny wpływać na jego bezpieczeństwo. Niestety – dalej jest to algorytm bardzo szybki.

Różnice między algorytmami z rodziny SHA przedstawia poniższa tabela:

Secure Hash Algorithm Family

Algorytm i wariant		Rozmiar wyjścia (bity)	Wewnętrzny rozmiar stanu (bity)	Max rozmiar wiadomości (bity)	Znalezione Kolizje	Przykładowa wydajność (MB/s)	
						32b	64b
SHA-0		160	160	$2^{64}-1$	Tak	-	-
SHA-1		160	160	$2^{64}-1$	Teoretyczny atak (2 ⁵¹)	153	192
SHA-2	SHA-256/224	256/224	256	$2^{64}-1$	Brak	111	139
	SHA-512/384	512/384	512	$2^{128}-1$	Brak	99	154

Rys. 2.3.1 SHA.

Poniżej przedstawiono obrazowy przykład działania programu ze strony <https://sha256algorithm.com> z wykorzystaniem tego sposobu haszowania (w wersji SHA-256) dla wygenerowania unikalnego hasza do hasła **AlaMaKota1234!**.

The screenshot displays the SHA-256 algorithm's internal state and message schedule. The input message 'AlaMaKota1234!' is shown at the top left. The main area is divided into three sections: 'Message schedule - 1st chunk', 'Working Variables', and 'K constants'. The 'Message schedule' section shows the iterative calculation of the hash value, starting with 'h = g' and 'g = f', and then calculating 'f = e + d + Temp1', 'd = c', 'c = b', 'b = a', and 'a = Temp1 + Temp2'. The 'Working Variables' section shows the state of the hash function, including 'right rotate 6', 'right rotate 11', 'right rotate 25', and 'Choice: f'. The 'K constants' section shows the values of the constants used in the algorithm, such as 'K1 = 0x428a2f98', 'K2 = 0xc5c01365', etc. The bottom section shows the final hash value 'Shaz56 = h0 h1 h2 h3 h4 h5 h6 h7'.

Rys. 2.3.2 SHA.^[3]

```

public class Program
{
    public static void Main1111()
    {
        string password = "test";
        using (SHA256 sha256Hash = SHA256.Create())
        {
            string hash = GetHash(sha256Hash, password);

            Console.WriteLine($"The SHA256 hash of '{password}' : {hash}.");

            Console.WriteLine("Verifying the hash...");

            if (VerifyHash(sha256Hash, password, hash))
            {
                Console.WriteLine("The hashes are the same.");
            }
            else
            {
                Console.WriteLine("The hashes are not same.");
            }
        }
        Console.ReadLine();
    }

    private static string GetHash(HashAlgorithm hashAlgorithm, string input)
    {
        byte[] data = hashAlgorithm.ComputeHash(Encoding.UTF8.GetBytes(input));

        var sBuilder = new StringBuilder();

        for (int i = 0; i < data.Length; i++)
        {
            sBuilder.Append(data[i].ToString("x2"));
        }

        return sBuilder.ToString();
    }

    private static bool VerifyHash(HashAlgorithm hashAlgorithm, string input, string hash)
    {
        var hashOfInput = GetHash(hashAlgorithm, input);

        StringComparer comparer = StringComparer.OrdinalIgnoreCase;

        return comparer.Compare(hashOfInput, hash) == 0;
    }
}

```

Rys. 2.3.3 Przykładowa implementacja SHA-256.

2.4. HASHOWANIE SOLONE.

Hash "solony" to technika używana w kryptografii i informatyce, która polega na dodaniu losowej wartości (tzw. "soli") do hasła przed jego haszowaniem. W procesie solenia, losowa wartość jest dodawana do hasła, a następnie całość jest poddawana funkcji skrótu (hash), takiej jak np. SHA-256. Wynikowy skrót jest zapisywany w bazie danych razem z wartością soli, która była użyta do solenia hasła. Dzięki temu, nawet jeśli dwa użytkownicy używają tego samego hasła, to ich solone hasła w bazie danych będą się od siebie różnić, ponieważ soli użyte do ich solenia będą różne. Dzięki temu, złamanie jednego hasła nie oznacza złamania wszystkich solonych haseł w bazie danych. Haszowanie solone jest jednym z podstawowych sposobów przechowywania haseł i jest stosowane w wielu systemach logowania i uwierzytelniania, w celu zwiększenia bezpieczeństwa przechowywanych haseł.

3. POSTĘPY W PROJEKCIE.

3.1. Wydajność Brute Force ?

Tempo łamania hasła zależy od wielu czynników, takich jak długość hasła, użyte znaki, algorytm szyfrowania, moc obliczeniowa używanego sprzętu, itp. Oczywiście, im dłuższe i bardziej złożone hasło, tym dłużej będzie trwać proces łamania za pomocą brute force. Dla przykładu, hasło składające się z 6 małych liter (26 możliwości na każdej pozycji) może zostać złamane w ciągu kilku minut na zwykłym komputerze. Natomiast, hasło składające się z 12 znaków (liczby, litery, znaki specjalne) może wymagać lat pracy dla najpotężniejszych superkomputerów.

```
password found: ~~~~ (4 characters)
It took: 0 day(s), 0h, 5min, 45s and 469ms

password found: ~~~~~ (5 characters)
It took: 0 day(s), 9h, 6min, 15s and 0ms

password found: ~~~~~ (6 characters)
It took: 36 day(s), 0h, 53min, 45s and 0ms

password found: ~~~~~ (7 characters)
It took: 3423 day(s), 13h, 6min, 15s and 0ms
(more than 9 years)

password found: ~~~~~ (8 characters)
It took: 325236 day(s), 20h, 53min, 45s and 0ms
(around 891 years)

password found: ~~~~~ (9 characters)
It took: 30897502 day(s), 17h, 6min, 15s and 0ms
(more than 84650 years)

password found: ~~~~~ (10 characters)
It took: 2935262757 day(s), 16h, 53min, 45s and 0ms
(around 8042 millennia)
```


3.2. Generowanie możliwych haseł na podstawie słownika.

Wariacje z powtórzeniami to matematyczna technika, która polega na wyznaczaniu różnych możliwych permutacji elementów, gdzie elementy te mogą się powtarzać. Oznacza to, że wariacje z powtórzeniami umożliwiają nam wybranie tego samego elementu więcej niż raz, co odróżnia je od wariacji bez powtórzeń, gdzie każdy element musi zostać wykorzystany tylko raz.

TWIERDZENIE

Liczba k -wyrazowych wariacji z powtórzeniami zbioru n -elementowego (ozn. W_n^k) wyraża się wzorem:

$$W_n^k = n^k \text{ gdzie } n, k \in \mathbb{N}_+$$

Rys. Wzór na wariacje z powtórzeniami wraz z twierdzeniem^[5]

Implementacja generowania wszystkich możliwych haseł na podstawie danego słownika przedstawia się następująco:

```
// This code was created out of inspiration by PrinciRaj1992 code (user from geeksforgeeks.org) - we adjust it to our project & add some functionality
public class GFG
{
    private static string path = Environment.GetFolderPath(Environment.SpecialFolder.Desktop).ToString() + "\\combinations" + ".txt";
    private static StringBuilder stringBuilder = new StringBuilder("");
    private static StringBuilder stringBuilderLocal = new StringBuilder("");

    static void allLexicographicRecur(String str, char[] data, int last, int index)
    {
        int length = str.Length;

        for (int i = 0; i < length; i++)
        {
            data[index] = str[i];

            if (index == last)
            {
                Console.WriteLine(new String(data));
                String s = new String(data) + "\n";
                stringBuilderLocal.Append(s);
            }
            else
            {
                allLexicographicRecur(str, data, last, index + 1);
            }
        }
    }

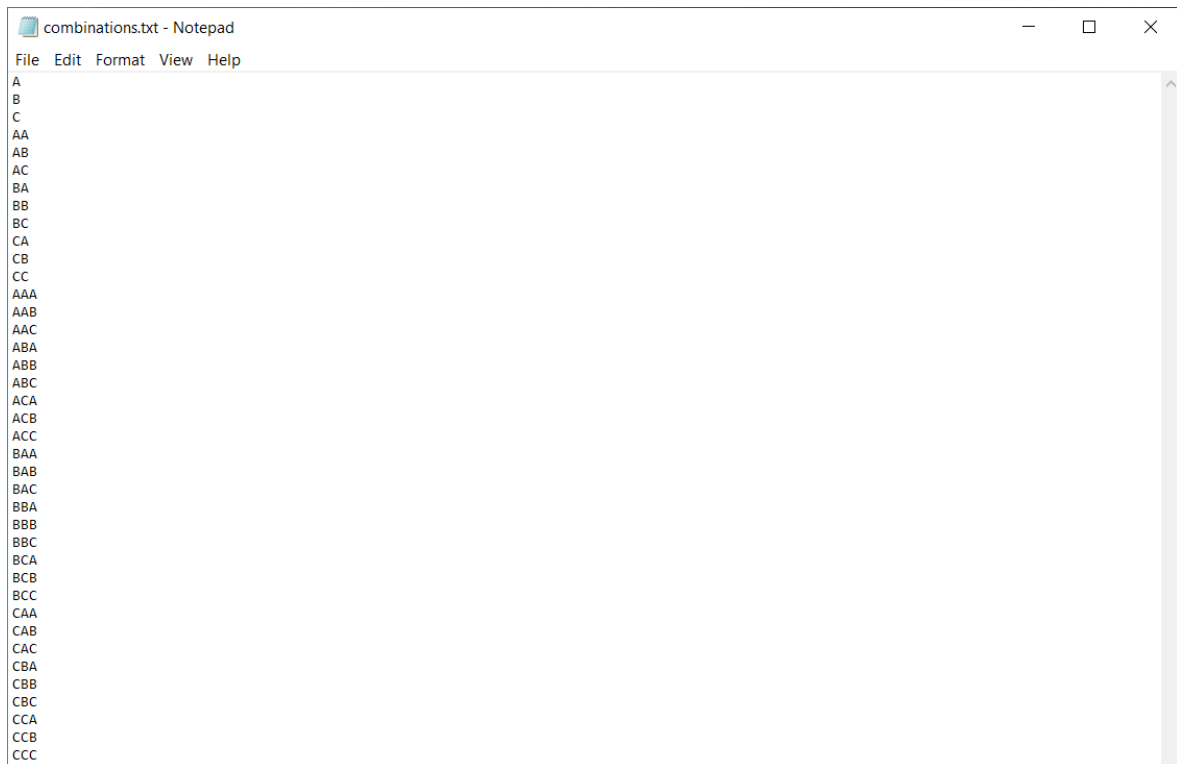
    public static void allLexicographic(String str, int minStrLength, int maxStrLength)
    {
        int maxLength = maxStrLength;

        for (int i = minStrLength - 1; i < maxStrLength; i++)
        {
            char[] data = new char[maxLength + 1];
            char[] temp = str.ToCharArray();

            Array.Sort(temp);
            str = new String(temp);

            allLexicographicRecur(str, data, i, 0);
            stringBuilder.Append(stringBuilderLocal.ToString());
            stringBuilderLocal = new StringBuilder("");
        }
        if (stringBuilder.ToString().EndsWith("\n"))
        {
            stringBuilder = stringBuilder.Remove(stringBuilder.Length - 2, 2);
        }

        SaveResultToFile(stringBuilder.ToString());
    }
}
```



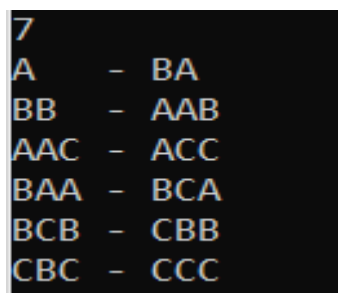
```
combinations.txt - Notepad
File Edit Format View Help
A
B
C
AA
AB
AC
BA
BB
BC
CA
CB
CC
AAA
AAB
AAC
ABA
ABB
ABC
ACA
ACB
ACC
BAA
BAB
BAC
BBA
BBB
BBC
BCA
BCB
BCC
CAA
CAB
CAC
CBA
CBB
CBC
CCA
CCB
CCC
```

Rys. Wyniki generowania możliwych kombinacji.

Funkcja została zaimplementowana pod kątem używania różnych długości generowanych ciągu znaków - można je ustawiać za pomocą odpowiednio zdefiniowanych parametrów (*minStrLength*, *maxStrLength*). Po wygenerowaniu wszystkich możliwych kombinacji, wynik zostaje zapisany do pliku tekstowego.

3.3. Wyznaczanie granic.

Aby obciążenie wszystkich komputerów w rozproszonej wersji projektu było równomierne potrzebowaliśmy odpowiedniej funkcji obliczającej oraz wyznaczającej granice wygenerowanych możliwych kombinacji, aby móc je rozsyłać do pozostałych komputerów (potencjalnych klientów).



```
7
A - BA
BB - AAB
AAC - ACC
BAA - BCA
BCB - CBB
CBC - CCC
```

Rys. Wyniki sugerowanych granic.

```

public static void GetBoundFromFile(int numberOfComputers)
{
    int numberOfFileLines = File.ReadLines(path).Count();
    int numberOfLines = numberOfFileLines / numberOfComputers;
    Console.WriteLine(numberOfLines + "");

    for (int i = 1; i < numberOfFileLines; i += numberOfLines)
    {
        if(i + numberOfLines - 1 > numberOfFileLines)
        {
            //Console.WriteLine(i + "-" + numberOfFileLines);
            Console.WriteLine(File.ReadLines(path).Skip(i - 1).Take(1).First() + "-" + File.ReadLines(path).Skip(numberOfFileLines - 1).Take(1).First());
        }
        else
        {
            //Console.WriteLine(i + "-" + (i + numberOfLines - 1));
            Console.WriteLine(File.ReadLines(path).Skip(i - 1).Take(1).First() + "-" + File.ReadLines(path).Skip(i + numberOfLines - 2).Take(1).First());
        }
    }
}

```

Rys. Wyniki generowania możliwych kombinacji.

3.4. Popularne hasła.

RockYou.txt to nazwa pliku tekstowego zawierającego ponad 32 miliony haseł, które zostały skradzione z serwisu społecznościowego RockYou w 2009 roku. RockYou był popularnym serwisem oferującym narzędzia do gier społecznościowych i aplikacji na Facebooku, MySpace i innych platformach. Jednak w grudniu 2009 roku serwis został zhakowany, a hakerzy ukradli dane logowania i hasła do ponad 32 milionów kont użytkowników. Hakerzy wykorzystali technikę ataku słownikowego, korzystając z dużej bazy haseł, w tym właśnie z pliku RockYou.txt. Plik ten zawierał ogromną liczbę haseł, które były używane przez użytkowników RockYou, a także przez innych użytkowników internetu na całym świecie. Hakerzy wykorzystali te hasła, aby uzyskać nieuprawniony dostęp do kont użytkowników RockYou oraz innych serwisów, na których ci użytkownicy mieli konta. RockYou.txt stał się jednym z największych i najbardziej popularnych plików z hasłami, wykorzystywanych przez hakerów na całym świecie. Plik ten stał się również jednym z najważniejszych narzędzi dla specjalistów ds. bezpieczeństwa w celu testowania zabezpieczeń systemów informatycznych i sieciowych. Dzięki RockYou.txt specjaliści mogą testować skuteczność swoich systemów zabezpieczeń i zapobiegać atakom hakerskim. Można go pobrać m.in z Githuba lub innych ogólnodostępnych stron.^[6]

LITERATURA

[1] Haszowanie

<https://bezpieczny.blog/co-to-jest-hash/>

[2] SHA

<https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona/#:~:text=SHA2%20istnieje%20w%20czterech%20odmianach,-384%20oraz%20SHA-512.>

[3] SHA

<https://sha256algorithm.com>

[4] Hashowanie

http://wojciech.bozejko.staff.iiar.pwr.wroc.pl/elearning/Wyk5_tabl_hash.pdf

[5] Twierdzenie wariacji z powtórzeniami

<https://opracowania.pl/opracowania/matematyka/wariacje-z-powtorzeniami,oid,2046>

[6] Opis rockyou.txt

<https://en.wikipedia.org/wiki/RockYou>